Основы программирования

Представление графов

Графы

Граф задается двумя множествами: вершин и ребер. Каждое ребро соединяет две вершины, т.е. может быть задано парой имен (номеров) вершин. Условно можно говорить, что ребро ab определяет возможность перехода из вершины a в вершину b.

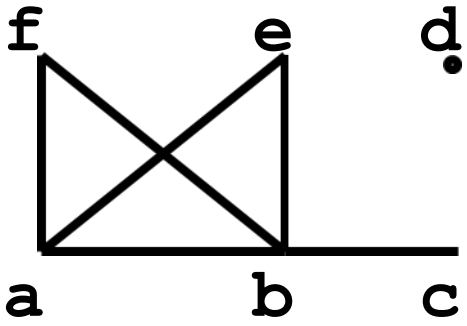
В неориентированном графе задание ребра ab определяет 2 возможных перехода: из a в b и из b в a.

В ориентированном графе задание дуги ab определяет только переход из a в b. Обратный переход возможен если задана также дуга ba.

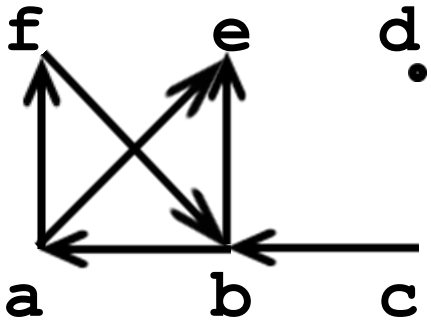
Графы часто представляют графически: точки (вершины) соединяют отрезками линий (ребрами) или стрелками (дугами ориентированного графа).

Графическое представление

Неориентированный



Ориентированный

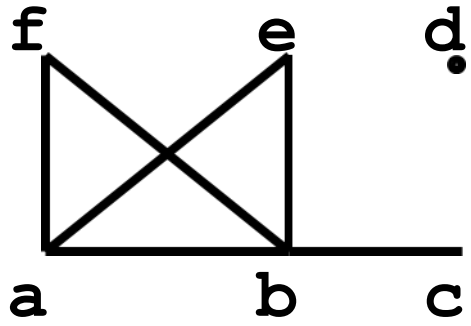


Существует несколько способов задания графа:

- матрица смежности определяет для всех пар вершин соединяются ли они ребрами (дугами)
- списки смежных вершин определяют для каждой вершины, с какими вершинами она связана
- матрица инцидентности определяет инцидентность всех вершин ребрам (используется очень редко)
- матрица весов – аналог матрицы смежности (вместо единиц и нулей задаются веса ребер
- массив ребер или дуг (массив пар вершин).

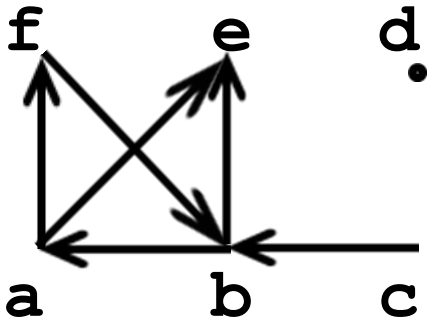
Матрицы смежности

Неориентированный



	a	b	c	d	e	f
a	0	1	0	0	1	1
b	1	0	1	0	1	1
c	0	1	0	0	0	0
d	0	0	0	0	0	0
e	1	1	0	0	0	0
f	1	1	0	0	0	0

Ориентированный

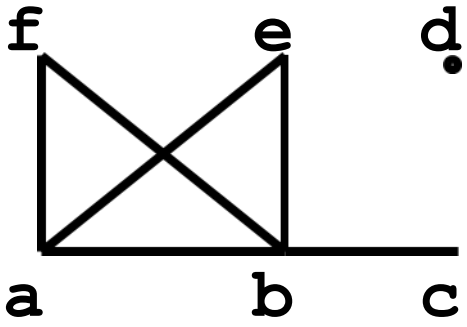


	a	b	c	d	e	f
a	0	0	0	0	1	1
b	1	0	0	0	1	0
c	0	1	0	0	0	0
d	0	0	0	0	0	0
e	0	0	0	0	0	0
f	0	1	0	0	0	0

Очевидно, что в программах используются не имена, а номера вершин графа.

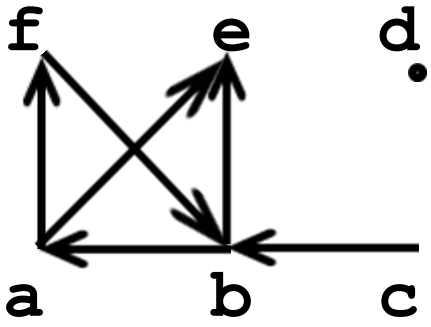
Списки смежных вершин

Неориентированный




```
a b-e-f
b a-c-e-f
c b
d
e a-b
f a-b
```

Ориентированный

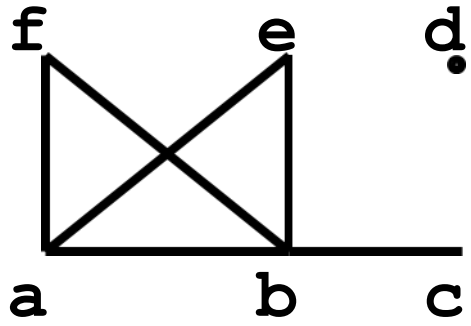



```
a e-f
b a-e
c b
d
e
f b
```

В отличие от матрицы смежности списки содержат только смежные вершины.

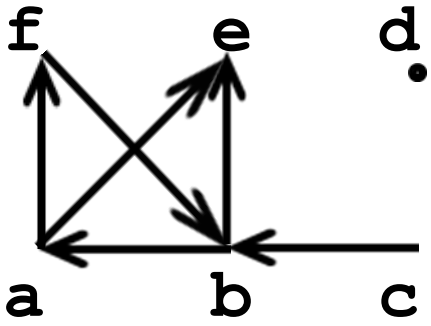
Матрицы инцидентности

Неориентированный



	a	b	c	d	e	f
ab	1	1	0	0	0	0
bc	0	1	1	0	0	0
ae	1	0	0	0	1	0
af	1	0	0	0	0	1
be	0	1	0	0	1	0
fb	0	1	0	0	0	1

Ориентированный

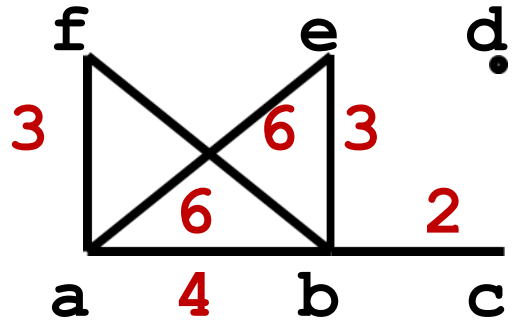


	a	b	c	d	e	f
ba	1	0	0	0	0	0
cb	0	0	1	0	0	0
ae	1	0	0	0	0	0
af	1	0	0	0	0	0
be	0	1	0	0	0	0
fb	0	0	0	0	0	1

В программах используются не имена, а номера вершин и ребер графа.

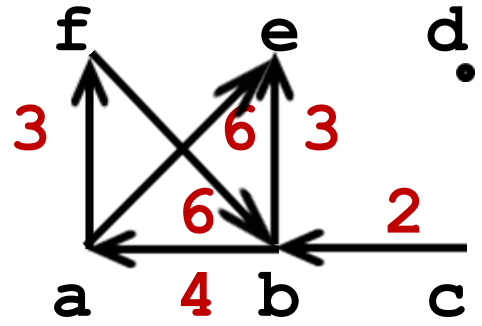
Матрицы весов

Неориентированный



	a	b	c	d	e	f
a	∞	4	∞	∞	6	3
b	4	∞	2	∞	3	6
c	∞	2	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞
e	6	3	∞	∞	∞	∞
f	3	6	∞	∞	∞	∞

Ориентированный

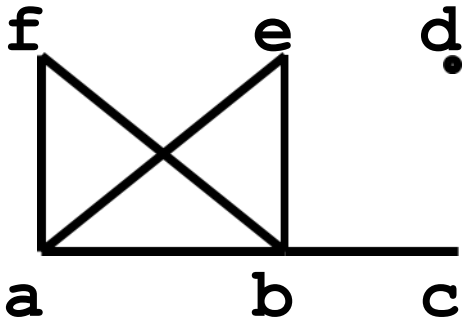


	a	b	c	d	e	f
a	∞	∞	∞	∞	6	3
b	4	∞	∞	∞	3	∞
c	∞	2	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞
e	∞	∞	∞	∞	∞	∞
f	∞	6	∞	∞	∞	∞

Бесконечные веса используются, т.к. обычно требуется найти объекты с минимальным весом (пути).

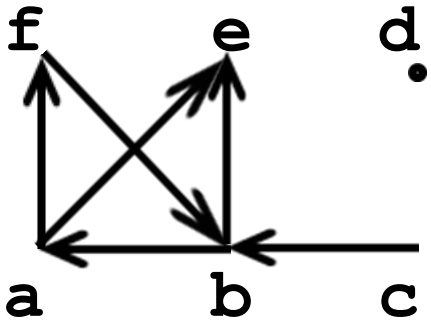
Массивы ребер (дуг)

Неориентированный




```
a b
b c
a f
a e
f b
b e
```

Ориентированный

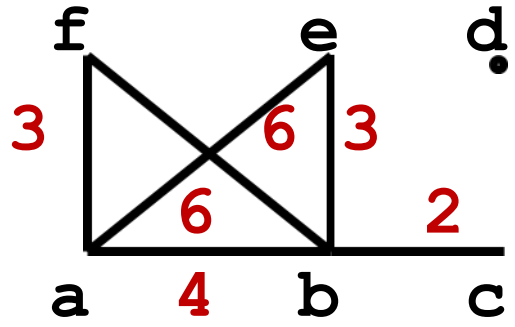



```
b a
c b
a f
a e
f b
b e
```

Массив ребер удобно использовать для ввода.

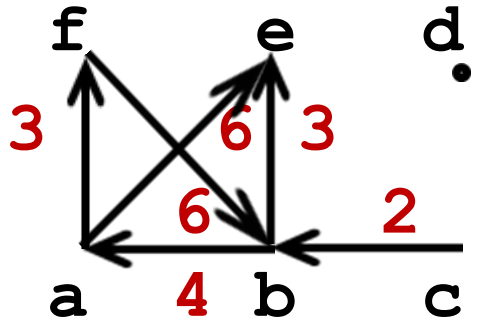
Массивы ребер (дуг) с весами

Неориентированный




```
a b 4
b c 2
a f 3
a e 6
f b 6
b e 3
```

Ориентированный




```
b a 4
c b 2
a f 3
a e 6
f b 6
b e 3
```

Списки смежных вершин взвешенного графа содержат пары «номер смежной вершины, вес ребра».

Классы для представления графов

Мы будем использовать 3 способа представления графа: матрицей смежности, списками смежных вершин и матрицей весов.

Для этого мы создадим, соответственно, классы **MGraph, LGraph, WGraph** с набором базовых методов и будем добавлять к ним дополнительные методы, реализующие некоторые алгоритмы на графах.

Класс MGraph

Класс для представления графа с помощью матрицы смежности:

```
class MGraph
{
    bool **mat;    // матрица смежности
    int vernum;    // число вершин
    bool oriented; // true - орграф
public:
    MGraph(int vnum, bool orient);
    ~MGraph();
    void input_edges();
    bool is_oriented() { return oriented; }
    int get_vernum() { return vernum; }
    bool is_edge(int a, int b);
};
```

Конструктор и деструктор MGraph

```
MGraph::MGraph(int vnum, bool orient)
{
    mat = new bool* [vnum];
    for (int i = 0; i < vnum; i++)
        mat[i] = new bool[n];
    vernum = vnum;
    oriented = orient;
}

MGraph::~MGraph()
{
    for (int i = 0; i < vernum; i++)
        delete [] mat[i];
    delete [] mat;
}
```

Ввод ребер (дуг) для MGraph

```
void MGraph::input_edges()
{
    int i, j;
    for (i = 0; i < vernum; i++)
        for (j = 0; j < vernum; j++)
            mat[i][j] = false;
    while (true)
    {
        cin >> i >> j;
        if (i < 0 || i >= vernum) return;
        if (j < 0 || j >= vernum) return;
        mat[i][j] = true;
        if (!oriented) mat[j][i] = true;
    }
}
```

Проверка существования ребра MGraph

```
bool MGraph::is_edge(int a, int b)
{
    if (a < 0 || a >= vernum) return false;
    if (b < 0 || b >= vernum) return false;
    return mat[a][b];
}
```

Класс LGraph

Класс для представления графа с помощью списков смежных вершин:

```
class LGraph
{
    List *lst;      // списки смежных вершин
    int vernum;     // число вершин
    bool oriented;  // true - орграф
public:
    LGraph(int vnum, bool orient);
    ~LGraph();
    void input_edges();
    bool is_oriented() { return oriented; }
    int get_vernum() { return vernum; }
    bool is_edge(int a, int b);
};
```

Конструктор и деструктор LGraph

```
LGraph::LGraph(int vnum, bool orient)
{
    lst = new List[vnum];
    vernum = vnum;
    oriented = orient;
}

LGraph::~LGraph()
{
    delete [] lst;
}
```

Ввод ребер (дуг) для LGraph

```
void LGraph::input_edges()
{
    int i, j;
    for (i = 0; i < vernum; i++)
        lst[i].clear();
    while (true)
    {
        cin >> i >> j;
        if (i < 0 || i >= vernum) return;
        if (j < 0 || j >= vernum) return;
        lst[i].push_back(j);
        if (!oriented) lst[j].push_back(i);
    }
}
```

Проверка существования ребра LGraph

```
bool LGraph::is_edge(int a, int b)
{
    if (a < 0 || a >= vernum) return false;
    if (b < 0 || b >= vernum) return false;
    if (lst[a].find(b) >= 0) return true;
    return false;
}
```

Класс WGraph

Класс для представления взвешенного графа с помощью матрицы весов:

```
#define INF 1e30
class WGraph
{   double **mat;// матрица весов
    int vernum;    // число вершин
    bool oriented;  // true - орграф
public:
    WGraph(int vnum, bool orient);
    ~WGraph();
    void input_edges();
    bool is_oriented() { return oriented; }
    int get_vernum() { return vernum; }
    double edge_weight(int a, int b);
};
```

Конструктор и деструктор WGraph

```
WGraph::WGraph(int vnum, bool orient)
{
    mat = new double* [vnum];
    for (int i = 0; i < vnum; i++)
        mat[i] = new double[n];
    vernum = vnum;
    oriented = orient;
}

WGraph::~WGraph()
{
    for (int i = 0; i < vernum; i++)
        delete [] mat[i];
    delete [] mat;
}
```

Ввод ребер (дуг) с весами для WGraph

```
void WGraph::input_edges()
{
    int i, j; double w;
    for (i = 0; i < vernum; i++)
        for (j = 0; j < vernum; j++)
            mat[i][j] = INF;
    while (true)
    {
        cin >> i >> j >> w;
        if (i < 0 || i >= vernum) return;
        if (j < 0 || j >= vernum) return;
        mat[i][j] = w;
        if (!oriented) mat[j][i] = w;
    }
}
```

Получение веса ребра WGraph

```
double WGraph::edge_weight(int a, int b)
{
    if (a < 0 || a >= vernum) return INF;
    if (b < 0 || b >= vernum) return INF;
    return mat[a][b];
}
```